An Overview of Ciao
(and its uses of DataLog in Program Analysis/Optimization)

Manuel Hermenegildo[4,1]

(with F. Bueno,[1] G. Puebla,[1] M. Carro,[1] P. López-García,[1] J. Morales,[2] E. Mera,[1] P. Chico,[1] J. Navas,[3] R. Haemmerlé,[1] M. Méndez,[3] A. Casas,[3] J. Correas,[2] E. Albert,[2] P. Arenas,[2])

The Ciao Development Team

[1]*CS Dept., T.U. of Madrid*

[2]*CS Dept., Complutense U. of Madrid*

[3]*CS and EECE Depts., U. of New Mexico, USA*

[4]*IMDEA Software Research Institute*

Datalog 2.0 – Oxford, March 19, 2010

Motivation and Approach

- Objectives:
  - Next-generation, high-level, *multiparadigm* programming language: Ciao.
  - *Program development environments which perform, as part of compilation*:
    - *Verification / debugging*
      (i.e., detect bugs and offer guarantees *of safety, reliability, and efficiency.)*
    - *Optimization (optimized compilation, parallelization, ...).*

    Using throughout techniques that are at the same time *rigorous* and *practical*.
  - Apply in a real system, with users –reality check!
  - Support also mainstream languages (e.g., Java / Java bytecode).
- Several uses of Datalog and related techniques.

Ciao Packages and Paradigms

- Built in layers over a small, LP-based *kernel*:
  - "Packages" provide *syntactic and semantic extensions and restrictions* on a per-module basis.
- *Logic programming:*
  - Certainly ISO-Prolog (one of the popular Prologs) –but *via a library*; and also:
  - Pure LP, *ASP (ASP-Prolog, Pontelli et al.), constructive negation,* ...
  - Various comp. rules: breadth-first, iterative-deepening, Andorra, *tabling*, etc.
- *Functional programming:*
  - Function definitions and function calls and functional syntax for predicates.
  - *Higher-order* and *lazyness* for functions and predicates.
- *Constraint programming:* clpr, clpq, fd, Leuven CHR.
- *Objects*.
- *Concurrency, parallelism, distributed execution*.
- *Assertion language*, consistent across paradigms; with many uses!

+ many other packages: type systems, records, *PiLLoW, RDF, XPath,* DSLs, ...

Some Uses of Datalog and Related Technology

- Intermediate representation for several Ciao analyses.
  - Groundness (mode) analysis: `def`, BDDs.
  - Definiteness propagation in analysis of (C)LP.
  - Assertion checking: comparator.
  - VC generation / checking in Abstraction Carrying Code.
  - Simplification of parallelization conditions.
- Also for Java: nullity, aliasing / sharing, resource usage.
- Simple solvers based generally on tabling.
- Bottom-up fixpoint evaluators also used in, e.g., shape / type inference.
  - Uses magic sets, etc.
- ASP modules (ASP-Prolog, Pontelli et al.).

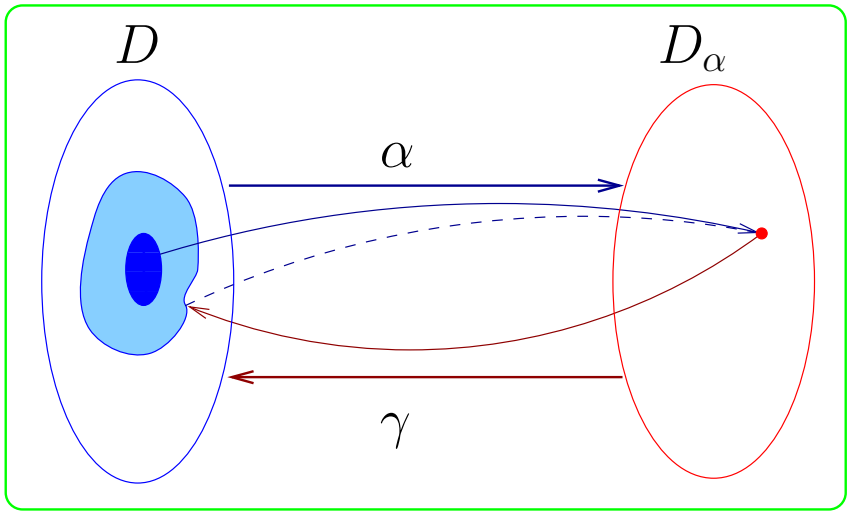
+ All tools written in Ciao (= Datalog++++ `:-)`).

Program Analysis and Optimization in Ciao

- Compute Safe over- and/or under-approximations of program semantics, $[\![P]\!]_{\alpha^+}$ or $[\![P]\!]_{\alpha^-}$
  generally based on *modular, polyvariant abstract interpretation*:

  $\forall x \in D:\ \gamma(\alpha(x)) \supseteq x$, and
  $\forall y \in D_\alpha:\ \alpha(\gamma(y)) = y$.

  $\mathrm{lfp}(S_P^\alpha) = [\![P]\!]_\alpha \supseteq \alpha([\![P]\!])$ terminates.



- Apply such approximations to verification, optimization.
- Domains: types, modes, pointer sharing, cost, sizes, termination, determinacy, non-fail, ...

Verification and Diagnosis via Abstract Interpretation [AADEBUG'97]

- Program verification/diagnosis: compare $[\![P]\!]$ with intended semantics $\mathcal{I}$ e.g.:

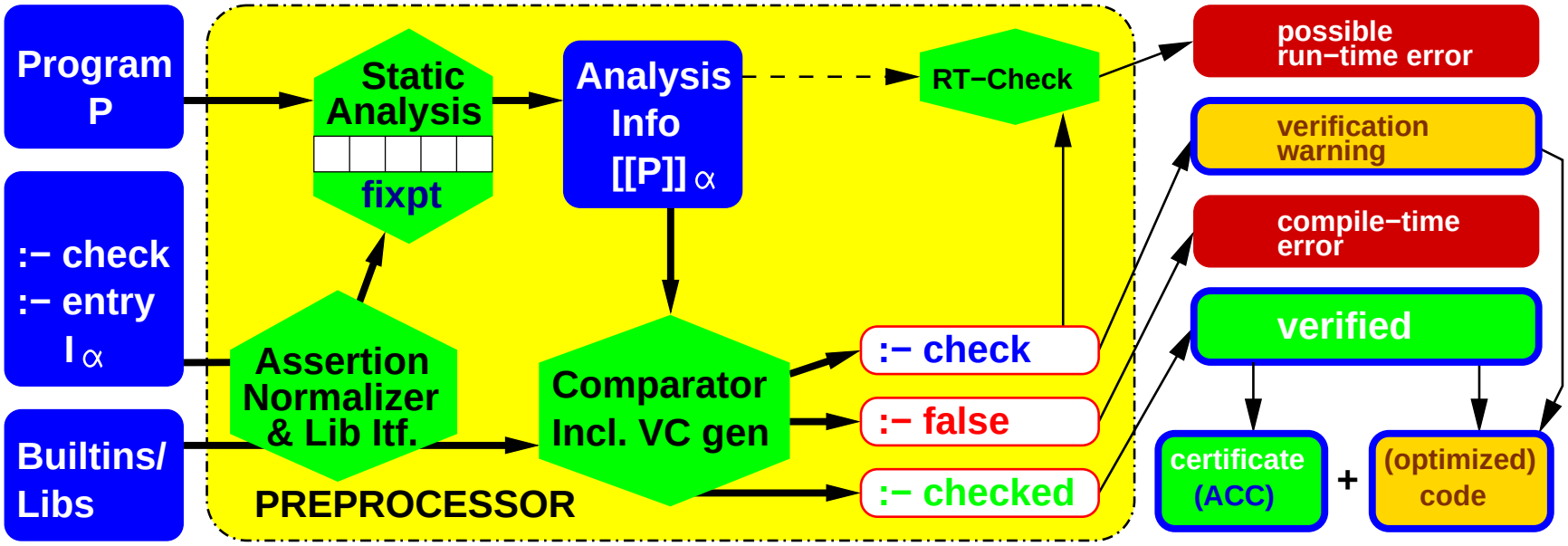
P is *partially correct* w.r.t. $\mathcal{I}$ iff	$[\![P]\!] \subseteq \mathcal{I}$
P is *complete* w.r.t. $\mathcal{I}$ iff	$\mathcal{I} \subseteq [\![P]\!]$
P is *incorrect* w.r.t. $\mathcal{I}$ iff	$[\![P]\!] \not\subseteq \mathcal{I}$
P is *incomplete* w.r.t. $\mathcal{I}$ iff	$\mathcal{I} \not\subseteq [\![P]\!]$

- Usually, only partial descriptions of $\mathcal{I}$ are available, typically as *assertions*.
- *Problem:* difficulty in computing $[\![P]\!] \rightarrow$ use *abstract interpretation* to compute a *safe approximation* $[\![P]\!]_{\alpha^+}$ $[\![P]\!]_{\alpha^+}$ indicates $[\![P]\!]_\alpha \supseteq \alpha([\![P]\!]$.

Property	**Definition**	**Sufficient condition**
P is partially correct w.r.t. $\mathcal{I}_\alpha$ if	$\alpha([\![P]\!]) \subseteq \mathcal{I}_\alpha$	$[\![P]\!]_{\alpha^+} \subseteq \mathcal{I}_\alpha$ if
P is complete w.r.t. $\mathcal{I}_\alpha$ if	$\mathcal{I}_\alpha \subseteq \alpha([\![P]\!])$	$\mathcal{I}_\alpha \subseteq [\![P]\!]_{\alpha^=}$
P is incorrect w.r.t. $\mathcal{I}_\alpha$ if	$\alpha([\![P]\!]) \not\subseteq \mathcal{I}_\alpha$	$[\![P]\!]_{\alpha^=} \not\subseteq \mathcal{I}_\alpha$, or $[\![P]\!]_{\alpha^+} \cap \mathcal{I}_\alpha = \emptyset \wedge [\![P]\!]_\alpha \neq \emptyset$
P is incomplete w.r.t. $\mathcal{I}_\alpha$ if	$\mathcal{I}_\alpha \not\subseteq \alpha([\![P]\!])$	$\mathcal{I}_\alpha \not\subseteq [\![P]\!]_{\alpha^+}$

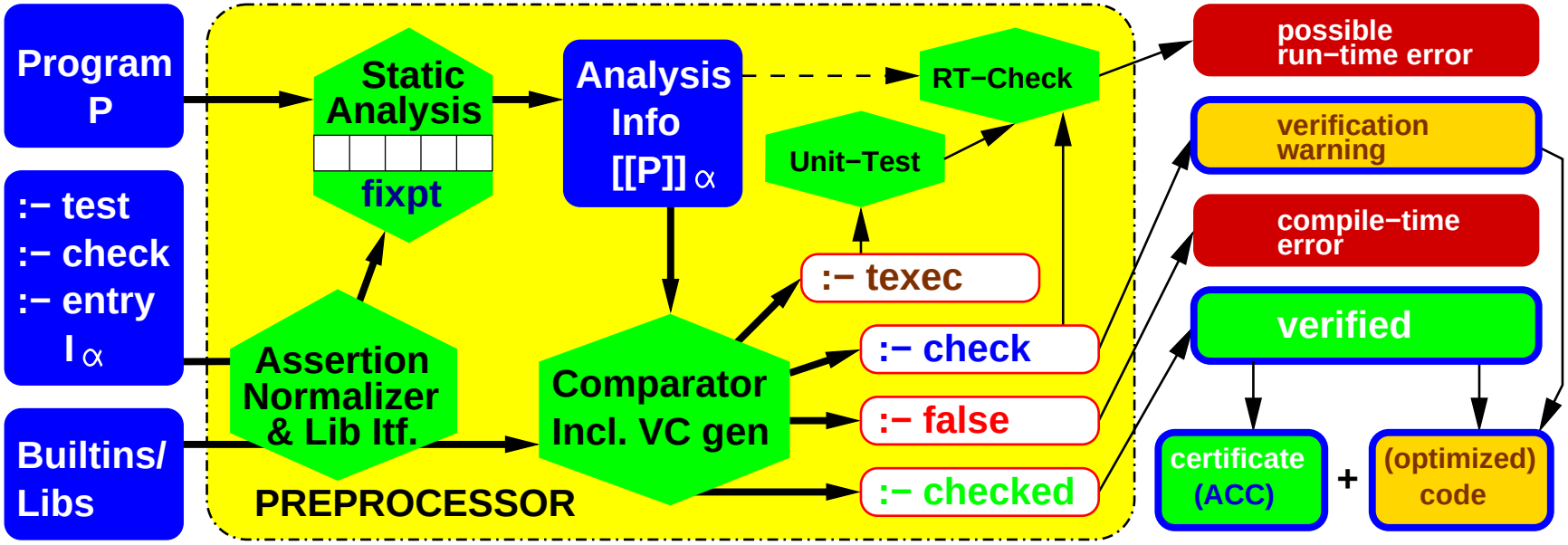
- Specially attractive if compiler computes (most of) $[\![P]\!]_{\alpha^+}$ anyway.

Verification/Diagnosis Framework (CiaoPP) [AADEBUG'97]



- I_α (partial spec.) provided via a language of *optional assertions*.
  - State properties at relevant point (pre, post, global, pp).
  - Talk about "properties," predefined or user-defined (in the source language).
- Types, modes, pointer sharing, cost, sizes, termination, determinacy, non-fail, ...

Verification/Diagnosis Framework (CiaoPP) [AADEBUG'97]



- I_α (partial spec.) provided via a language of *optional assertions*.
  - State properties at relevant point (pre, post, global, pp).
  - Talk about "properties," predefined or user-defined (in the source language).
- Types, modes, pointer sharing, cost, sizes, termination, determinacy, non-fail, ...

Assertion Language: Properties

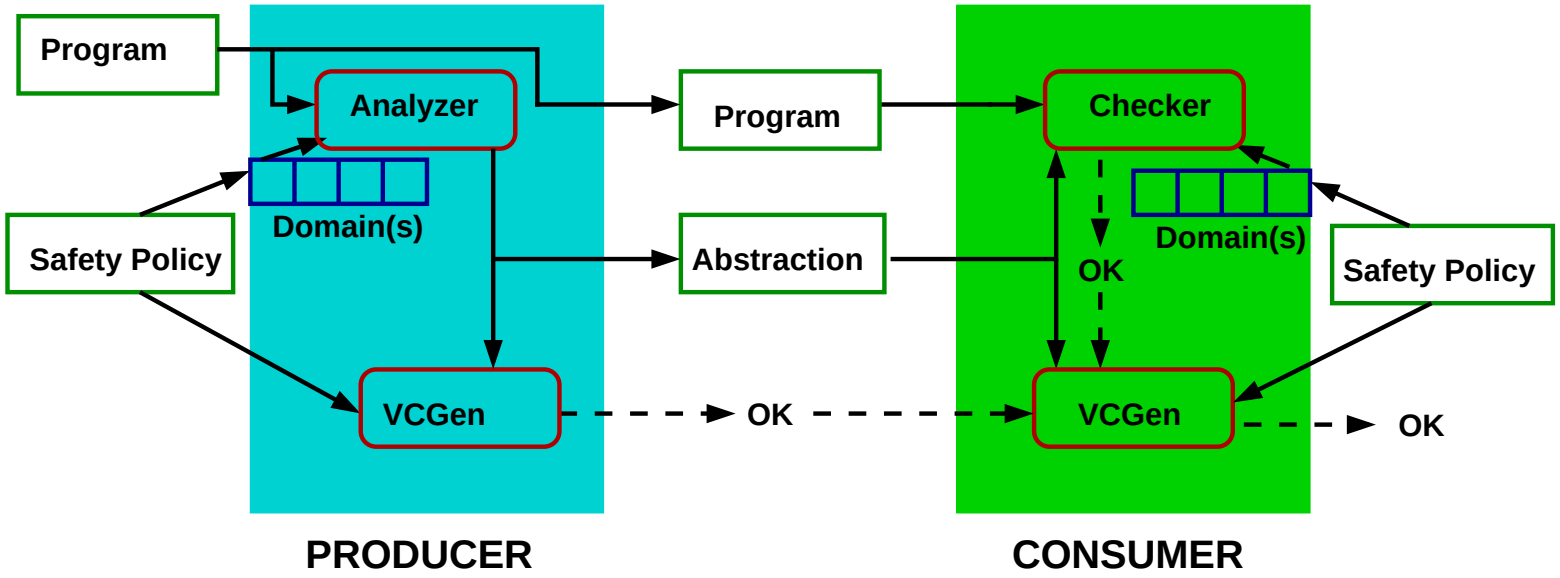
```
:- regtype list/1.                          | :- regtype list/1.
list([]).                                   | list :=  [] | [_|~list].
list([_|Y]) :- list(Y).                     |_____________________________
____________________________________________| :- regtype color/1.
:- prop sorted/1.                           | color := green | blue | red.
sorted([]).                                 |_____________________________
sorted([_]).                                | :- regtype peano_int/1.
sorted([X,Y|Z]) :- X>Y, sorted([Y|Z]).      | peano_int := 0 | s(~peano_int).
```

- Arbitrary predicates in restricted logic (a subset of Ciao).
- Some conditions on them: termination, no instantiation, ...
- Many predefined in system libs, some of them “native” to an analyzer.
- Can also be user-defined.
- Should be visible in the module and “runnable:” they will be used also as run-time tests! (but the property may be an approximation itself).
- *Types* are a special case of property (e.g., `regtype`s).
- But also, e.g., argument sizes, instantiation states, sizes, cost, ...

Assertion Language: *Pred* Assertions

- `:- pred` *PredPattern* `[ :` *Pre* `] [ =>` *Post* `] [ +` *Comp*`].`
  - *Closed* on calls: cover all uses of a predicate (they imply a calls assertion).
  - Assertion *status*: `check`, `true`/`false`, `trust`, `checked`.
- Some examples, and some syntactic sugar:
  - `:- pred qsort(X,Y) => sorted(Y).`
  - `:- pred qsort(X,Y) : list(int) * var => sorted(Y) + (is_det,not_fails).`
    `:- pred qsort(X,Y) : var * list(int)) => ground(X) + not_fails.`
  - `:- pred foo(X,Y) : ground * var => (ground(Y), X>Y) + det.`
    `:- pred foo(X,Y) : var * ground => (ground(X), X>Y).`
  - `:- trust pred is/2 => num * numexpr.`
  - `:- modedef +X : nonvar(X).`
  - `:- pred sortints(+L,-SL) :: list(int) * list(int) => sorted(SL)`
    `# "@var{SL} has same elements as @var{L}.".`

The Abstraction Carrying Code (ACC) Scheme [COCV04,LPAR04]



$[\![P]\!]_\alpha = \text{Analysis} = \text{lfp}(analysis\_step)$ | $\text{Certificate} \subset [\![P]\!]_\alpha$ | $\text{Checker} = analysis\_step$

- Scheme incorporated in `CiaoPP`, with domains: types, modes, data structure shape (including pointer sharing), bounds on data structure sizes, determinacy, termination, non-failure, bounds on resource consumption (time or space cost), ...

Big discussion (90's :-)): comparison with "classical" Types

- Allows going well beyond the "straight-jacket" of classical type systems:

"Traditional" Types	**CiaoPP Assertion-based Model [AADEBUG'97]**
"Property" language limited by decidability	Much more general property language
May need to limit programming language	No need to limit programming language
"Untypable" programs rejected	Run-time checks introduced
(Almost) Decidable	Decidable and Undecidable (and Approximated)
Expressed in a special language	(Expressed in the source language –for LP)
Types must be defined	Types can be defined or inferred
Assertions are only of type "check"	"check", "trust", ...
Type signatures and assertions different	Type signatures *are* assertions

...without giving up much (types are included as just another kind of property).

- Some key issues:

Approximation — *Suitable assertion language*

Abstract Interpretation — *Powerful abstract domains*

- Worst best when properties and assertions can be expressed in the source language (i.e., the source language supports *predicates* and *constraints*).

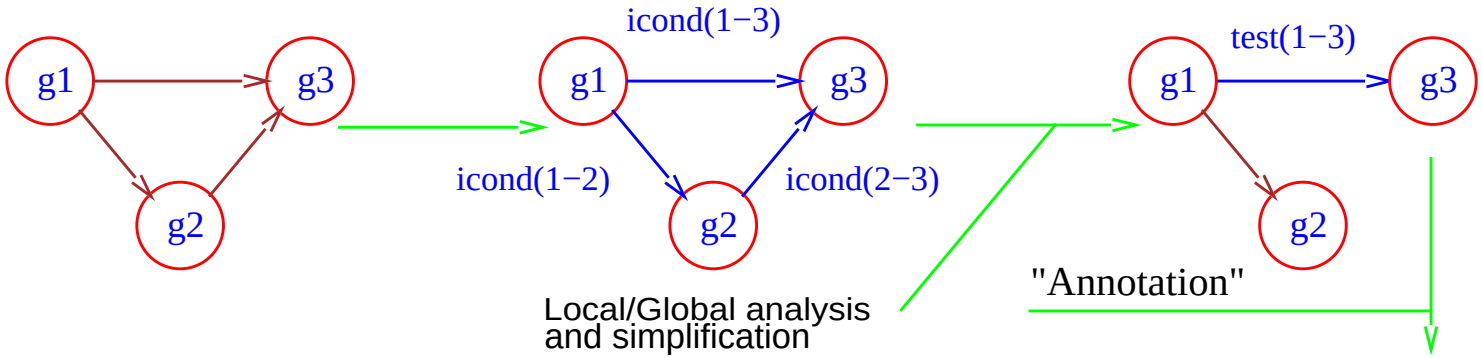
Also, optimizations

- Preprocessor architecture useful not just for verification / debugging, but also for optimization:
  - Source-level optimizations:
    - Partial evaluation, (multiple) (abstract) specialization, slicing, ...
  - Low-level (WAM) optimizations:
    - Use of specialized instructions.
    - Optimized native code generation.
  - **Parallelization.**

Automatic Program Parallelization

- Parallelization process starts with dependency graph:
  - edges exist if there can be a dependency,
  - conditions label edges if the dependency can be removed.
- Annotation: encoding of parallelism in the target parallel language:

$g_1(...)$, $g_2(...)$, $g_3(...)$



Alternative: g1, (g2 & g3)

- Global *sharing/aliasing* analysis: reduce/eliminate checks in conditions.
- Granularity control: based on cost / size analysis.

Automatic Program Parallelization (Contd.)

- *Example:*

```
qs([X|L],R) :- part(L,X,L1,L2),
               qs(L2,R2), qs(L1,R1),
               app(R1,[X|R2],R).
```

Might be annotated in &-Prolog (or Ciao Prolog), using local analysis, as:

```
qs([X|L],R) :-
         part(L,X,L1,L2),
         ( indep(L1,L2) ->
              qs(L2,R2) & qs(L1,R1)
         ;    qs(L2,R2) , qs(L1,R1) ),
         app(R1,[X|R2],R).
```

Global analysis would eliminate the `indep(L1,L2)` check.

Other Related Recent Work

- Framework adapted to Java and Java bytecode (Mobius):
  - Developed specific framework for Java/Java bytecode .
  - Analysis/validation of Java bytecode via P.Eval. of interpreter .
- Scalability, modularity .
- ACC:
  - Reduced certificates .
  - Incremental ACC (and other advanced PCC scenarios) .
- Extension of cost analysis to *time* bounds .
- Cost analysis of Java bytecode
- Extension to generic user-defined resources (Prolog and Java).
  - Examples: bytes sent over Internet, files open, DB accesses, etc.
- Abstract diagnosis.

(+ IMDEA-Software Development Technology Institute!)

Some Members of The Ciao Forge

- Ciao is really a widely distributed collaborative effort:
  - Directly within the CLIP Group:

    M. Hermenegildo, K. Muthukumar, M. García de la Banda, F. Bueno, G. Puebla, M. Carro, D. Cabeza, P. López-G., E. Albert, J. Navas, P. Chico, A. Casas, J. Correas, R. Haemmerlé, M. Méndez, J. Morales, E. Mera, C. Ochoa, P. Pietrzak, P. Arenas, S. Genaim

  - Plus lots of contributors worldwide:

    G. Gupta (UT Dallas), E. Pontelli (NM State University), P. Stuckey and M. García de la Banda (Melbourne U.), K. Marriott (Monash U.), M. Bruynooghe, A. Mulkers, G. Janssens, and V. Dumortier (K.U. Leuven), S. Debray (U. of Arizona), J. Maluzynski and W. Drabent, (Linkoping U.), P. Deransart (INRIA), J. Gallagher (Roskilde University), C. Holzbauer (Austrian Research Institute for AI), M. Codish (Beer-Sheva), SICS, ...

Some Selected Bibliography on the Ciao System

All papers can be found on line at: `http://clip.dia.fi.upm.es/clippubsbyyear` and `http://clip.dia.fi.upm.es/clippubsbytopic`

- System manual:

[1] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, and G. Puebla (Eds.). The Ciao System. Ref. Manual (v1.13). Technical report, C. S. School (UPM), 2006. Available at `http://www.ciaohome.org`.

- Overall design and philosophy:

[1] M. V. Hermenegildo, F. Bueno, M. Carro, P. López and J.F. Morales, and G. Puebla. An Overview of The Ciao Multiparadigm Language and Program Development Environment and its Design Philosophy. *Festschrift for Ugo Montanari*. June 2008, LNCS 5065, pages 209–237, Springer-Verlag.

[2] M. Hermenegildo, E. Albert, P. López-García, and G. Puebla. Some Techniques for Automated, Resource-Aware Distributed and Mobile Computing in a Multi-Paradigm Programming System. In *Proc. of EURO–PAR 2004*, number 3149 in LNCS, pages 21–37. Springer-Verlag, August 2004.

[3] D. Cabeza. *An Extensible, Global Analysis Friendly Logic Programming System*. PhD thesis, Universidad Politécnica de Madrid (UPM), Facultad Informatica UPM, 28660-Boadilla del Monte, Madrid-Spain, August 2004.

[4] M. Hermenegildo, F. Bueno, D. Cabeza, M. Carro, M. García de la Banda, P. López-García, and G. Puebla. The Ciao Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems. In *Parallelism and Implementation of Logic and Constraint Logic Programming*, pages 65–85. Nova Science, Commack, NY, USA, April 1999.

[5] M. Hermenegildo and The CLIP Group. Some Methodological Issues in the Design of Ciao - A Generic, Parallel, Concurrent Constraint System. In *Principles and Practice of Constraint Programming*, number 874 in LNCS, pages 123–133. Springer-Verlag, May 1994.

Functions, higher order, lazyness:

[1] A. Casas, D. Cabeza, and M. Hermenegildo. A Syntactic Approach to Combining Functional Notation, Lazy Evaluation and Higher-Order in LP Systems. In *Eighth International Symposium on Functional and Logic Programming (FLOPS'06)*, Fuji Susono (Japan), April 2006.

[2] D. Cabeza, M. Hermenegildo, and J. Lipton. Hiord: A Type-Free Higher-Order Logic Programming Language with Predicate Abstraction. In *Ninth Asian Computing Science Conference (ASIAN'04)*, number 3321 in LNCS, pages 93–108. Springer-Verlag, December 2004.

Tabling:

[1] P. Chico de Guzmán, M. Carro, M. Hermenegildo, Claudio Silva, Ricardo Rocha. An Improved Continuation Call-Based Implementation of Tabling. *10th International Symposium on Practical Aspects of Declarative Languages (PADL'08)*, LNCS, Vol. 4902, pages 198-213, Springer-Verlag, January 2008.

[2] Pablo Chico de Guzmán Huerta, Manuel Carro, Manuel Hermenegildo. Towards a Complete Scheme for Tabled Execution Based on Program Transformation. *11th International Symposium on Practical Aspects of Declarative Languages (PADL'09)*, LNCS, Num. 5418, pages 224-238, Springer-Verlag, January 2009.

Objects:

[1] A. Pineda and F. Bueno. The O'Ciao Approach to Object Oriented Logic Programming. In *Colloquium on Implementation of Constraint and LOgic Programming Systems (ICLP associated workshop)*, Copenhagen, July 2002.

[2] M. Carro and M. Hermenegildo. A simple approach to distributed objects in prolog. In *Colloquium on Implementation of Constraint and LOgic Programming Systems (ICLP associated workshop)*, Copenhagen, July 2002.

Auto-documenter:

[1] M. Hermenegildo. A Documentation Generator for (C)LP Systems. In *International Conference on Computational Logic, CL2000*, number 1861 in LNAI, pages 1345–1361. Springer-Verlag, July 2000.

Asbtract machine and low-level optimization:

[1] A. Casas, M. Carro, M. Hermenegildo. A High-Level Implementation of Non-Deterministic, Unrestricted, Independent And-Parallelism. *24th International Conference on Logic Programming (ICLP'08)*, LNCS, Vol. 5366, pages 651-666, Springer-Verlag, December 2008.

[2] M. Carro, J. Morales, H.L. Muller, G. Puebla, and M. Hermenegildo. High-Level Languages for Small Devices: A Case Study. In Krisztian Flautner and Taewhan Kim, editors, *Compilers, Architecture, and Synthesis for Embedded Systems*, pages 271–281. ACM Press / Sheridan, October 2006.

[3] J. Morales, M. Carro, G. Puebla, and M. Hermenegildo. A generator of efficient abstract machine implementations and its application to emulator minimization. In P. Meseguer and J. Larrosa, editors, *International Conference on Logic Programming*, number 3668 in LNCS, pages 21–36. Springer Verlag, October 2005.

[4] J.F. Morales, M. Carro, and M. Hermenegildo. Towards Description and Optimization of Abstract Machines in an Extension of Prolog. In *Logic-Based Program Synthesis and Transformation (LOPSTR'06)*, number 4407 in LNCS, pages 77–93, July 2007.

[5] J. Morales, M. Carro, M. Hermenegildo. Comparing Tag Scheme Variations Using an Abstract Machine Generator. *10th Int'l. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'08)*, pages 32-43, ACM Press, July 2008.

[6] J. Morales, M. Carro, and M. Hermenegildo. Improving the Compilation of Prolog to C Using Moded Types and Determinism Information. In *Proceedings of the Sixth International Symposium on Practical Aspects of Declarative Languages*, number 3057 in LNCS, pages 86–103, Heidelberg, Germany, June 2004. Springer-Verlag.

Automatic parallelization:

[1] D. Cabeza, M. Hermenegildo. Non-Strict Independence-Based Program Parallelization Using Sharing and Freeness Information. *Theoretical Computer Science*, Vol. 46, Num. 410, pages 4704-4723, Elsevier Science, October 2009.

[2] M. Hermenegildo, F. Bueno, A. Casas, J. Navas, E. Mera, M. Carro, and P. López-García. Automatic Granularity-Aware Parallelization of Programs with Predicates, Functions, and Constraints. In *DAMP'07, ACM SIGPLAN Workshop on Declarative Aspects of Multicore Programming*, January 2007.

[3] A. Casas, M. Carro, M. Hermenegildo. Annotation Algorithms for Unrestricted Independent And-Parallelism in Logic Programs. *17th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'07)*, LNCS, Num. 4915, pages 138-153, Springer-Verlag, August 2007.

[4] A. Casas, M. Carro, and M. Hermenegildo. Annotation Algorithms for Unrestricted Independent And-Parallelism in Logic Programs. In *17th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'07)*, The Technical University of Denmark, August 2007. Springer-Verlag.

[5] M. Hermenegildo. Automatic Parallelization of Irregular and Pointer-Based Computations: Perspectives from Logic and Constraint Programming. In *Proceedings of EUROPAR'97*, volume 1300 of *LNCS*, pages 31–46. Springer-Verlag, August 1997. Invited.

[6] F. Bueno, M. García de la Banda, and M. Hermenegildo. Effectiveness of Abstract Interpretation in Automatic Parallelization: A Case Study in Logic Programming. *ACM Transactions on Programming Languages and Systems*, 21(2):189–238, March 1999.

[7] K. Muthukumar, F. Bueno, M. García de la Banda, and M. Hermenegildo. Automatic Compile-time Parallelization of Logic Programs for Restricted, Goal-level, Independent And-parallelism. *Journal of Logic Programming*, 38(2):165–218, February 1999.

[8] D. Cabeza and M. Hermenegildo. Extracting Non-strict Independent And-parallelism Using Sharing and Freeness Information. In *1994 International Static Analysis Symposium*, number 864 in LNCS, pages 297–313, Namur, Belgium, September 1994. Springer-Verlag.

[9] M. Hermenegildo and K. Greene. The &-Prolog System: Exploiting Independent And-Parallelism. *New Generation Computing*, 9(3,4):233–257, 1991.

- Cost analysis and granularity control in parallelism:

[1] S. K. Debray, N.-W. Lin, and M. Hermenegildo. Task Granularity Analysis in Logic Programs. In *Proc. of the 1990 ACM Conf. on Programming Language Design and Implementation*, pages 174–188. ACM Press, June 1990.

[2] S.K. Debray, P. López-García, M. Hermenegildo, and N.-W. Lin. Estimating the Computational Cost of Logic Programs. In *Static Analysis Symposium, SAS'94*, number 864 in LNCS, pages 255–265, Namur, Belgium, September 1994. Springer-Verlag.

[3] P. López-García, M. Hermenegildo, and S. K. Debray. A Methodology for Granularity Based Control of Parallelism in Logic Programs. *Journal of Symbolic Computation, Special Issue on Parallel Symbolic Computation*, 21(4–6):715–734, 1996.

[4] E. Mera, P. López-García, G. Puebla, M. Carro, and M. Hermenegildo. Combining Static Analysis and Profiling for Estimating Execution Times. In *Ninth International Symposium on Practical Aspects of Declarative Languages*, number 4354 in LNCS, pages 140–154. Springer-Verlag, January 2007.

[5] J. Navas, E. Mera, P. López-García, and M. Hermenegildo. User-Definable Resource Bounds Analysis for Logic Programs. In *23rd International Conference on Logic Programming (ICLP 2007)*, LNCS. Springer-Verlag, September 2007.

[6] E. Mera, P. López-García, M. Carro, M. Hermenegildo. Towards Execution Time Estimation in Abstract Machine-Based Languages. *10th Int'l. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'08)*, pages 174-184, ACM Press, July 2008.

- The overall program development framework (CiaoPP):

[1] F. Bueno, P. Deransart, W. Drabent, G. Ferrand, M. Hermenegildo, J. Maluszynski, and G. Puebla. On the Role of Semantic Approximations in Validation and Diagnosis of Constraint Logic Programs. In *Proc. of the 3rd. Int'l Workshop on Automated Debugging–AADEBUG'97*, pages 155–170, Linköping, Sweden, May 1997. U. of Linköping Press.

[2] M. Hermenegildo, G. Puebla, and F. Bueno. Using Global Analysis, Partial Specifications, and an Extensible Assertion Language for Program Validation and Debugging. In K. R. Apt, V. Marek, M. Truszczynski, and D. S. Warren, editors, *The Logic Programming Paradigm: a 25–Year Perspective*, pages 161–192. Springer-Verlag, July 1999.

[3] G. Puebla, F. Bueno, and M. Hermenegildo. Combined Static and Dynamic Assertion-Based Debugging of Constraint Logic Programs. In *Logic-based Program Synthesis and Transformation (LOPSTR'99)*, number 1817 in LNCS, pages 273–292. Springer-Verlag, March 2000.

[4] G. Puebla, F. Bueno, and M. Hermenegildo. A Generic Preprocessor for Program Validation and Debugging. In P. Deransart, M. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 63–107. Springer-Verlag, September 2000.

[5] G. Puebla, F. Bueno, and M. Hermenegildo. An Assertion Language for Constraint Logic Programs. In P. Deransart, M. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 23–61. Springer-Verlag, September 2000.

[6] M. Hermenegildo, G. Puebla, F. Bueno, and P. López-García. Abstract Verification and Debugging of Constraint Logic Programs. In *Recent Advances in Constraints*, number 2627 in LNCS, pages 1–14. Springer-Verlag, January 2003.

[7] M. Hermenegildo, G. Puebla, F. Bueno, and P. López-García. Program Development Using Abstract Interpretation (and The Ciao System Preprocessor). Invited talk. In *10th International Static Analysis Symposium (SAS'03)*, number 2694 in LNCS, pages 127–152. Springer-Verlag, June 2003.

[8] E. Mera, P. López-García, M. Hermenegildo. Integrating Software Testing and Run-Time Checking in an Assertion Verification Framework. em International Conference on Logic Programming (ICLP), LNCS, Num. 5649, pages 281-295, Springer-Verlag, July 2009.

Abstraction carrying code:

[1] E. Albert, G. Puebla, and M. Hermenegildo. Abstraction-Carrying Code. In *Proc. of LPAR'04*, number 3452 in LNAI, pages 380–397. Springer-Verlag, 2005.

[2] E. Albert, G. Puebla, and M. Hermenegildo. An Abstract Interpretation-based Approach to Mobile Code Safety. In *Proc. of Compiler Optimization meets Compiler Verification (COCV'04)*, Electronic Notes in Theoretical Computer Science 132(1), pages 113–129. Elsevier - North Holland, April 2004.

[3] E. Albert, P. Arenas, G. Puebla, and M. Hermenegildo. Reduced Certificates for Abstraction-Carrying Code. In *22nd International Conference on Logic Programming (ICLP 2006)*, number 4079 in LNCS, pages 163–178. Springer-Verlag, August 2006.

[4] E. Albert, P. Arenas, and G. Puebla. An Incremental Approach to Abstraction-Carrying Code. In *13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'06)*, number 4246 in LNCS, pages 377–391. Springer-Verlag, November 2006.

Partial evaluation:

[1] G. Puebla and M. Hermenegildo. Abstract Specialization and its Applications. In *ACM Partial Evaluation and Semantics based Program Manipulation (PEPM'03)*, pages 29–43. ACM Press, June 2003. Invited.

[2] E. Albert, G. Puebla, and J. Gallagher. Non-Leftmost Unfolding in Partial Evaluation of Logic Programs with Impure Predicates. In *15th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'05)*, number 3901 in LNCS, pages 115–132. Springer-Verlag, April 2006.

[3] G. Puebla and C. Ochoa. Poly-Controlled Partial Evaluation. In *Proc. of 8th ACM-SIGPLAN International Symposium on Principles and Practice of Declarative Programming (PPDP'06)*, pages 261–271. ACM Press, July 2006.

[4] G. Puebla, E. Albert, and M. Hermenegildo. Abstract Interpretation with Specialized Definitions. In *The 13th International Static Analysis Symposium (SAS'06)*, number 4134 in LNCS, pages 107–126. Springer, August 2006.

[5] G. Puebla and M. Hermenegildo. Abstract Multiple Specialization and its Application to Program Parallelization. *J. of Logic Programming. Special Issue on Synthesis, Transformation and Analysis of Logic Programs*, 41(2&3):279–316, November 1999.

Scalability, modularity of analysis, debugging, and verification:

[1] P. Pietrzak, J. Correas, G. Puebla, and M. Hermenegildo. Context-Sensitive Multivariant Assertion Checking in Modular Programs. In *13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'06)*, number 4246 in LNCS, pages 392–406. Springer-Verlag, November 2006.

[2] F. Bueno, M. García de la Banda, M. Hermenegildo, K. Marriott, G. Puebla, and P. Stuckey. A Model for Inter-module Analysis and Optimizing Compilation. In *Logic-based Program Synthesis and Transformation*, number 2042 in LNCS, pages 86–102. Springer-Verlag, March 2001.

[3] G. Puebla, J. Correas, M. Hermenegildo, F. Bueno, M. García de la Banda, K. Marriott, and P. J. Stuckey. A Generic Framework for Context-Sensitive Analysis of Modular Programs. In M. Bruynooghe and K. Lau, editors, *Program Development in Computational Logic, A Decade of Research Advances in Logic-Based Program Development*, number 3049 in LNCS, pages 234–261. Springer-Verlag, Heidelberg, Germany, August 2004.

[4] P. Pietrzak, J. Correas, G. Puebla, M. Hermenegildo. A Practical Type Analysis for Verification of Modular Prolog Programs. *ACM SIGPLAN 2008 Workshop on Partial Evaluation and Program Manipulation (PEPM'08)*, pages 61-70, ACM Press, January 2008.

Some applications of the CiaoPP framework to Java bytecode:

[1] M. Méndez-Lojo, M. Hermenegildo. Precise Set Sharing Analysis for Java-style Programs. *9th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'08)*, LNCS, Num. 4905, pages 172-187, Springer-Verlag, January 2008.

[2] M. Méndez-Lojo, J. Navas, and M. Hermenegildo. A Flexible (C)LP-Based Approach to the Analysis of Object-Oriented Programs. In *17th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'07)*, August 2007.

[3] J. Navas, M. Méndez-Lojo, and M. Hermenegildo. An Efficient, Context and Path Sensitive Analysis Framework for Java Programs. In *9th Workshop on Formal Techniques for Java-like Programs FTfJP 2007*, July 2007.

[4] E. Albert, M. Gómez-Zamalloa, L. Hubert, and G. Puebla. Verification of Java Bytecode using Analysis and Transformation of Logic Programs. In *Ninth International Symposium on Practical Aspects of Declarative Languages*, number 4354 in LNCS, pages 124–139. Springer-Verlag, January 2007.